Звіт
Лабораторна робота 1
ОБ’ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ
Большаков Андрій МІТ-31
https://github.com/Utilka/OOP_labs_Univ

Тема :
Знайомство з классами та об’єктами у Python

Мета:
Ознайомитися з парадигмою об’єктно-орієнтованого програмування, розглянути поняття класу та об’єкта, навчитися створювати класи та об’єкти в Пайтон.

11. ЗАВДАННЯ 1 (3 бали у разі повного і правильного виконання). Самостійно опишіть клас згідно свого варіанту завдання (таблиця 1.1), створіть декілька екземплярів класу, продемонструйте функціонування усіх описаних у ньому методів. ЗВЕРНІТЬ УВАГУ, що у класі повинні бути хоча б одна статична змінна та статичний метод. Поясніть їх особливості та призначення.

```
class Safe:

    counter = 0

    def __init__(self, password):
        """
        create new instance of safe
        increases safe counter
        by default state is set to Open (True)


        :type password: object
        """
        self.password = password
        self.opened_state = True
        Safe.counter += 1

    def open(self, password):
        """
        open safe, change state to  Open (True)
        password argument must match password stored in safe's memory

        :type password: object
        """
        if (self.password == password):
            self.opened_state = True
        else:
            raise Exception("cant open safe: incorrect password")

    def close(self):
        """close safe, change state to  Closed (False)"""
        self.opened_state = False
```

```
    def set_password(self, new_password):
        """
        set new password for safe
        will only work if this safe is opened

        :type new_password: object
        """
        if (self.opened_state == True):
            self.password = new_password
        else:
            raise Exception("cant change password: safe is not opened")

    @staticmethod
    def beep():
        """this static method is here only for the sake of the assignment"""
        return "beep-boop"


if __name__ == '__main__':

    print(f"Safe.counter          : {Safe.counter}")
    safes = [Safe("123")]
    print(f"Safe.counter          : {Safe.counter}")
    safes = safes + [Safe("1234"),Safe("12345")]
    print(f"Safe.counter          : {Safe.counter}")

    safes[0].close()
    print(f"Safe_0.opened_state   : {safes[0].opened_state}")
    print(f"Safe_2.opened_state   : {safes[2].opened_state}")
    safes[0].open("123")
    print(f"Safe_0.opened_state   : {safes[0].opened_state}")
    safes[0].set_password("qwerty")
    print(f"Safe_0.password       : {safes[0].password}")
    print(f"Safe_2.password       : {safes[2].password}")
```

```
$ python3 main.py
Safe.counter          : 0
Safe.counter          : 1
Safe.counter          : 3
Safe_0.opened_state   : False
Safe_2.opened_state   : True
Safe_0.opened_state   : True
Safe_0.password       : qwerty
Safe_2.password       : 12345
```

12. ЗАВДАННЯ 2 (3 бали у разі повного і правильного виконання). Напишіть аналогічний клас іншою об'єктно-орієнтованою мовою програмування на ваш вибір, порівняйте створені реалізації та проведіть аналіз знайдених відмінностей. Поясніть одержані результати.

```
class Safe
  attr_accessor :counter, :opened, :password
  @@counter = 0

  # create new instance of safe
  # increases safe counter
  # by default state is set to Open (True)
```

```ruby
  # @param [Object] password
  def initialize(password)
    @password = password
    @opened = true
    @@counter += 1
  end

  # open safe, change state to Open (True)
  # password argument must match password stored in safe's memory
  # @param [Object] password
  def open(password)
    if @password == password
      @opened = true
    else
      raise StandardError.new("cant open safe: incorrect password")
    end
  end

  # close safe, change state to  Closed (False)
  def close()
    @opened = true
  end

  # set new password for safe
  # will only work if this safe is opened
  # @param [Object] new_password
  def set_password(new_password)
    if @opened
      @password = new_password
    else
      raise StandardError.new("cant change password : safe is not opened")

    end
  end

  # this static method is here only for the sake of the assignment
  def self.beep()
    "" "" ""
    return "beep-boop"
  end

  def self.counter
    @@counter
  end
end

if __FILE__ == $0
  puts("Safe.counter          : #{Safe.counter}")
  safes = [Safe.new("123")]
  puts("Safe.counter          : #{Safe.counter}")
  safes = safes + [Safe.new("1234"), Safe.new("12345")]
  puts("Safe.counter          : #{Safe.counter}")

  safes[0].close()
  puts("Safe_0.opened_state    : #{safes[0].opened}")
  puts("Safe_2.opened_state    : #{safes[2].opened}")
  safes[0].open("123")
  puts("Safe_0.opened_state    : #{safes[0].opened}")
  safes[0].set_password("qwerty")
  puts("Safe_0.password        : #{safes[0].password}")
  puts("Safe_2.password        : #{safes[2].password}")
```

```
end
$ ruby main.rb
Safe.counter           : 0
Safe.counter           : 1
Safe.counter           : 3
Safe_0.opened_state    : false
Safe_2.opened_state    : true
Safe_0.opened_state    : true
Safe_0.password        : qwerty
Safe_2.password        : 12345
```

Касательно руби
Я пришел к выводу что это нетипичный язык программирования, синтаксис заметно отличается от классики ООП C# и Java, и даже несколько отличается от питона, но в этом задании этого не так видно, чувствуется что под несколько другим синтаксисом прячется заметно другой зверь, в котором я еще не разобрался.

По ООП, самое заметное отличие это инкапсуляция, все переменные являются приватными, и для того чтобы получить к ним доступ из вне нужно прописать геттеры сеттеры, для создания иллюзии прозрачности можно создать методы с точно таким же названием как и название переменной.
Наследование в руби разрешено только одинарное, но можно достичь чегото похожего на множественное наследование, с помощью миксинов.
Полиморфизм явных изменений не претерпел как мне кажется.

Висновок:
Я ознайомився з парадигмою об’єктно-орієнтованого програмування, розглянув поняття класу та об’єкта, навчитися створювати класи та об’єкти в Пайтон.